Domande 2a prova (informatica)

⬜ Esame di Stato informatica:

- Diagramma ER, Logico e fisico (con vincoli di dominio e referenziali, indici)

- Pagine WEB (mockup, codice client, codice server e query)

⬜ Quesiti:

(Sempre presente) (Modifica del diagramma ER/tabelle precedente)

(Tema 18-19 ordinaria) (ruolo nei DB)

Nell’interazione con un’applicazione web dinamica, l’utente compie azioni che richiedono l’invio di dati al server. Il candidato esamini i metodi attraverso cui è possibile trasferire al server i dati generati lato client dall’utente durante

l’uso dell’applicazione, evidenziandone le specificità e i differenti usi. Fornisca al riguardo esempi di casi di utilizzo per le differenti modalità.

1. Modulo HTML e POST: Un modo comune per inviare dati al server è utilizzare un modulo HTML e il metodo di invio POST. I dati inseriti dall'utente all'interno dei campi di input del modulo vengono inviati al server quando l'utente invia il modulo. Questi dati possono essere utilizzati per elaborare l'input dell'utente o aggiornare lo stato dell'applicazione. Ad esempio, un modulo di registrazione che richiede all'utente di inserire nome utente e password può inviare questi dati al server tramite una richiesta POST per creare un nuovo account utente.
2. Query string e GET: I dati possono essere anche trasmessi al server tramite la stringa di query nell'URL utilizzando il metodo GET. In questo caso, i dati sono inclusi nell'URL stesso dopo il simbolo "?". Ad esempio, un'applicazione di ricerca potrebbe consentire all'utente di inserire una parola chiave e quindi includere quella parola chiave nella stringa di query per inviare la richiesta di ricerca al server. Il server può quindi elaborare la richiesta e restituire i risultati appropriati.
3. XMLHttpRequest e AJAX: Un'altra modalità per inviare dati al server in modo asincrono è utilizzare la tecnologia XMLHttpRequest (XHR) insieme alla tecnica di programmazione asincrona JavaScript e AJAX (Asynchronous JavaScript and XML). Questo metodo consente di inviare richieste HTTP al server senza dover ricaricare l'intera pagina. Gli sviluppatori possono utilizzare XHR per inviare dati al server e ricevere una risposta senza interrompere l'esperienza dell'utente. Ad esempio, in un'applicazione di chat in tempo reale, l'invio di un messaggio al server utilizzando XHR consente all'utente di continuare a vedere e inviare messaggi senza dover ricaricare l'intera pagina.
4. Fetch API: La Fetch API è una nuova API JavaScript che fornisce un'interfaccia per effettuare richieste HTTP. È una soluzione moderna e più potente rispetto all'XMLHttpRequest, offrendo una sintassi più semplice e promesse per una gestione più efficiente dei dati inviati e ricevuti dal server. Con Fetch API, è possibile inviare dati al server in vari formati, come JSON o FormData. Ad esempio, un'applicazione di e-commerce potrebbe utilizzare Fetch API per inviare i dettagli dell'ordine al server per il checkout.

(Tema 16-17 ordinaria)

In relazione al tema proposto nella prima parte, il candidato immagini di volere documentare al committente l’operatività della piattaforma proposta. A tal fine, imposti una relazione tecnica che presenti le principali caratteristiche dell’applicazione Web in termini di organizzazione e funzionalità. In particolare, imposti la struttura di tale relazione, motivando le scelte e scrivendo un esempio significativo dei relativi contenuti.

Relazione Tecnica sull'Organizzazione e le Funzionalità dell'Applicazione Web

Introduzione:
Nella presente relazione tecnica, verranno presentate le principali caratteristiche dell'applicazione web proposta in termini di organizzazione e funzionalità. L'obiettivo è fornire una panoramica completa dell'applicazione e illustrare come essa soddisfi le esigenze del committente.

1. Descrizione dell'Applicazione:
   In questa sezione, verrà fornita una panoramica generale dell'applicazione web, compresa una descrizione del suo scopo e delle sue funzionalità principali. Saranno illustrate le tecnologie utilizzate per lo sviluppo, l'architettura dell'applicazione e il modo in cui i vari componenti interagiscono tra loro.
   - Esempio:
     L'applicazione web proposta è un sistema di gestione degli ordini per un negozio online. Permette ai clienti di visualizzare i prodotti disponibili, aggiungerli al carrello e completare l'acquisto. Inoltre, offre un'interfaccia di

amministrazione per i gestori del negozio per gestire i prodotti, i clienti e gli ordini. L'applicazione è sviluppata utilizzando il framework JavaScript React per il front-end e Node.js per il back-end. L'architettura è basata su un'API RESTful che consente la comunicazione tra il client e il server.

2. Struttura dell'Applicazione:
   Questa sezione descrive l'organizzazione e la struttura dell'applicazione web. Saranno presentati i componenti principali, come le pagine, i moduli, i servizi e le librerie utilizzate. Verranno spiegate le dipendenze tra i vari componenti e l'organizzazione dei file e delle cartelle nel progetto.
   - Esempio:
     L'applicazione è organizzata in diversi componenti React riutilizzabili. Le pagine dell'applicazione corrispondono alle diverse sezioni dell'interfaccia utente, come la homepage, la pagina dei prodotti e la pagina del carrello. Ogni pagina è costituita da componenti più piccoli che gestiscono specifiche funzionalità. Ad esempio, il componente "ProductList" si occupa di visualizzare la lista dei prodotti e il componente "Cart" gestisce il carrello dell'utente. I componenti comunicano tra loro utilizzando le props e possono utilizzare servizi per accedere ai dati dal server.
3. Funzionalità dell'Applicazione:
   Questa sezione elenca le principali funzionalità dell'applicazione web e spiega come vengono implementate. Verranno forniti esempi di casi d'uso e sarà descritto il flusso di lavoro dell'utente durante l'utilizzo dell'applicazione.
   - Esempio:
     Le principali funzionalità dell'applicazione includono:
     - Registrazione e accesso degli utenti: Gli utenti possono creare un account o accedere utilizzando le loro credenziali.
     - Visualizzazione dei prodotti: Gli utenti possono navigare tra i prodotti disponibili e visualizzarne i dettagli, come immagini, descrizione e prezzo.
     - Aggiunta al carrello: Gli utenti possono aggiungere i prodotti desiderati al carrello e specificare la quantità.
     - Gestione del carrello: Gli utenti possono visualizzare il contenuto del carrello, modificare la quantità dei prodotti e rimuoverli.
     - Checkout: Gli utenti possono completare l'acquisto, fornendo le informazioni di spedizione e il metodo di pagamento.
     - Gestione degli ordini: I gestori del negozio possono visualizzare e gestire gli ordini ricevuti, aggiornare lo stato di consegna e generare report.

Conclusioni:
In questa sezione, vengono riassunte le principali caratteristiche dell'applicazione web, evidenziando come esse soddisfino le esigenze del committente. Verranno anche menzionate eventuali sviluppi futuri o possibili miglioramenti per l'applicazione.

La presente relazione ha fornito un'ampia panoramica dell'applicazione web proposta, illustrando la sua organizzazione, le funzionalità e il modo in cui si integra con le tecnologie e gli strumenti utilizzati. Questa documentazione tecnica servirà come riferimento per comprendere l'operatività dell'applicazione e facilitare il suo sviluppo e la sua manutenzione.

(Tema 16-17 ordinaria)

Dato il seguente schema relazionale:

- film (id, titolo, durata, anno di produzione, genere);

- attore (id, nome, cognome, data_nascita, fotografia);
- recita (id_film, id_attore, ruolo);

il candidato:

- determini la modalità di gestione del campo ‘fotografia’ che prevede la memorizzazione di una immagine dell’attore in un formato grafico (es. JPG);
- formalizzi in linguaggio SQL lo schema fisico corrispondente allo schema relazionale, sapendo che:
  - il campo ‘genere’ ammette solo i seguenti valori: fantasy, giallo, commedia, horror, drammatico, fantascienza, azione; b. per la relazione ‘recita’, i campi ‘id_film’ e ‘id_attore’ referenziano rispettivamente la chiave primaria delle relazioni ‘film’ e ‘attore’;
  - discuta l’uso degli indici nel modello fisico di una base di dati e suggerisca con motivato giudizio indici appropriati per questo schema relazionale, definendoli in linguaggio SQL.

Modalità di gestione del campo 'fotografia':
Per gestire il campo 'fotografia', che contiene un'immagine dell'attore in un formato grafico come JPG, è possibile utilizzare la seguente modalità:

- Per gestire il campo 'fotografia' che memorizza un'immagine dell'attore in un formato grafico come il JPG, è possibile adottare diverse modalità di gestione:
  1. Memorizzazione diretta: In questo caso, l'immagine viene convertita in un formato binario e memorizzata direttamente nel campo 'fotografia' del database. Il campo dovrebbe essere di tipo BLOB (Binary Large Object) o VARBINARY per consentire la memorizzazione di dati binari.
  2. Memorizzazione del percorso del file: In alternativa alla memorizzazione diretta dell'immagine nel database, è possibile memorizzare solo il percorso del file immagine sul server. Il campo 'fotografia' conterrà quindi la stringa del percorso del file (ad esempio, "/path/to/image.jpg").

     La scelta tra queste modalità dipende dalle esigenze specifiche dell'applicazione. La memorizzazione diretta è più adatta quando l'immagine deve essere facilmente accessibile e gestita direttamente dal database. D'altra parte, la memorizzazione del percorso del file può essere preferibile quando l'immagine è di grandi dimensioni o quando è necessario gestire le immagini in modo indipendente dal database.
- Schema fisico corrispondente allo schema relazionale:

```
CREATE TABLE film (
id INT PRIMARY KEY,
titolo VARCHAR(255),
durata INT,
anno_di_produzione INT,
genere VARCHAR(255) CHECK (genere IN ('fantasy', 'giallo', 'commedia', 'horror', 'drammatico', 'fantascienza', 'azione'))
);
```

```
CREATE TABLE attore (
id INT PRIMARY KEY,
nome VARCHAR(255),
cognome VARCHAR(255),
data_nascita DATE,
fotografia VARCHAR(255)
);
```

```
CREATE TABLE recita (
id_film INT,
id_attore INT,
ruolo VARCHAR(255),
FOREIGN KEY (id_film) REFERENCES film (id),
FOREIGN KEY (id_attore) REFERENCES attore (id)
);
```

- Uso degli indici nel modello fisico di una base di dati:
  Gli indici sono strutture di dati che migliorano le prestazioni delle interrogazioni sul database, accelerando la ricerca e il recupero dei dati. Possono essere utilizzati per ridurre il tempo di esecuzione delle query e migliorare l'efficienza complessiva del sistema.

  Per lo schema relazionale proposto, possiamo considerare gli indici seguenti:

  ```
  Indice sulla colonna 'id' della tabella 'film':
  CREATE INDEX idx_film_id ON film (id);
  ```

  ```
  Indice sulla colonna 'id' della tabella 'attore':
  CREATE INDEX idx_attore_id ON attore (id);
  ```

  ```
  Indice composito sulla colonna 'id_film' e 'id_attore' della tabella 'recita':
  CREATE INDEX idx_recita_id_film_id_attore ON recita (id_film, id_attore);
  ```

  Questi indici aiuteranno ad accelerare le interrogazioni che coinvolgono le chiavi primarie e straniere, consentendo un accesso più rapido ai dati corrispondenti. Tuttavia, la scelta degli indici dipende anche dalle interrogazioni specifiche che vengono eseguite più frequentemente. È importante monitorare le prestazioni del sistema e aggiornare o creare nuovi indici in base alle esigenze effettive.

(Tema 16-17 ordinaria)

Un’azienda desidera sviluppare un’applicazione Web per la prenotazione on-line di eventi culturali, fruibile sia da computer desktop che da dispositivi mobili come tablet e smartphone. Il candidato esponga i punti critici da affrontare relativamente alle differenti proprietà di visualizzazione delle varie tipologie di dispositivi e alla rispettiva fruizione dei contenuti. Illustri possibili misure risolutive, con esempi relativi all’applicazione in questione.

- Punti critici relativi alle proprietà di visualizzazione dei dispositivi mobili:
  1. Dimensioni dello schermo: I dispositivi mobili, come smartphone e tablet, hanno schermi di dimensioni ridotte rispetto ai computer desktop. Ciò significa che i contenuti dell'applicazione devono adattarsi in modo appropriato per garantire una buona esperienza utente su schermi più piccoli.
  2. Orientamento dello schermo: I dispositivi mobili possono essere utilizzati sia in modalità orizzontale che verticale. L'applicazione deve essere in grado di gestire entrambi gli orientamenti e adattare i contenuti di conseguenza.
  3. Input tattile: I dispositivi mobili utilizzano input tattili, come toccare e trascinare, anziché il mouse. L'applicazione deve essere progettata per supportare l'interazione touch e fornire elementi di navigazione e pulsanti di dimensioni adeguate per una facile selezione.

4. Connessione di rete: La connessione di rete sui dispositivi mobili può variare in termini di velocità e stabilità. L'applicazione deve essere progettata per funzionare in modo affidabile anche con connessioni di rete più lente o intermittenti.

- Misure risolutive:

1. Responsive Design: Utilizzare il responsive design per garantire che l'applicazione si adatti automaticamente alle dimensioni dello schermo. Ciò può essere realizzato utilizzando media query CSS per applicare stili diversi in base alle dimensioni dello schermo del dispositivo.
    - Esempio di meta tag e media query nel file HTML:

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
    /* Stili per schermi di dimensioni ridotte */
    @media (max-width: 767px) {
      /* Stili specifici per dispositivi mobili */
    }

    /* Stili per schermi di dimensioni più grandi */
    @media (min-width: 768px) {
      /* Stili specifici per computer desktop */
    }
  </style>
</head>
<body>
  <!-- Contenuti dell'applicazione -->
</body>
</html>
```

2. Touch-friendly UI: Progettare l'interfaccia utente dell'applicazione in modo che sia facile da utilizzare con input tattili. Utilizzare pulsanti di dimensioni adeguate, fornire spazi vuoti sufficienti per evitare errori di selezione accidentale e utilizzare gesti intuitivi come scorrimento e pinch-to-zoom.
3. Ottimizzazione delle immagini: Ridimensionare e comprimere le immagini per ridurre il tempo di caricamento dell'applicazione su connessioni di rete più lente. Utilizzare formati di immagine leggeri come JPEG o WebP e specificare dimensioni fisse o relative in modo che le immagini si adattino al layout in modo appropriato.
4. Gestione delle connessioni di rete: Implementare un feedback appropriato per gli utenti in caso di connessione di rete debole o assente. Ad esempio, visualizzare un messaggio di avviso o utilizzare indicatori di caricamento per informare l'utente sullo stato dell'operazione in corso.

Queste sono solo alcune delle possibili misure risolutive per affrontare le sfide legate alla visualizzazione e alla fruizione dei contenuti su dispositivi mobili. È importante testare l'applicazione su diversi dispositivi e monitorare le metriche di utilizzo per identificare eventuali problemi e apportare le modifiche necessarie.

(Tema 16-17 suppletiva)

In relazione al tema sviluppato nella prima parte, il candidato integri la base di dati in modo da tenere conto delle differenti tipologie di soggetti che possono interagire con essa: a. studenti/genitori, che devono visualizzare solo le proprie assenze; b. docenti, che possono visualizzare e inserire le assenze; c. personale di segreteria, che può inserire i dati relativi a nuovi studenti e visualizzare le assenze di tutti gli studenti; e sviluppi, con appropriati linguaggi a scelta sia lato client che lato server, il codice necessario per visualizzare un menu che offra le sole funzioni significative per il profilo dell'utente accreditato.

Per integrare la base di dati al fine di tener conto delle diverse tipologie di soggetti che interagiscono con essa, è possibile aggiungere una tabella 'Utenti' che conterrà le informazioni relative ai diversi profili utente, come studenti, genitori, docenti e personale di segreteria. Inoltre, verrà aggiunta una colonna 'ruolo' nella tabella degli utenti per identificare il ruolo di ciascun utente.

- Schema modificato della base di dati:

```
CREATE TABLE utenti (
id INT PRIMARY KEY,
nome VARCHAR(255),
cognome VARCHAR(255),
ruolo VARCHAR(255)
);
```

```
CREATE TABLE assenze (
id INT PRIMARY KEY,
id_utente INT,
data_assenza DATE,
motivo VARCHAR(255),
FOREIGN KEY (id_utente) REFERENCES utenti (id)
);
```

In base al ruolo degli utenti, l'applicazione dovrà mostrare diverse funzionalità. Di seguito viene fornito un esempio di codice lato client e lato server per visualizzare un menu con le sole funzioni significative per il profilo dell'utente accreditato.

- Esempio di codice lato server (PHP):

```
<?php
// Codice per verificare l'utente loggato e ottenere il suo ruolo dalla base di dati
// $userRole contiene il ruolo dell'utente loggato

// Generazione del menu in base al ruolo del'utente
$menu = '';

if ($userRole == 'studente' || $userRole == 'genitore') {
  // Funzioni per studenti/genitori
  $menu .= '<a href="#">Visualizza le mie assenze</a>';
}

if ($userRole == 'docente') {
  // Funzioni per docenti
  $menu .= '<a href="#">Visualizza le assenze</a>';
  $menu .= '<a href="#">Inserisci un'assenza</a>';
}

if ($userRole == 'segreteria') {
  // Funzioni per personale di segreteria
  $menu .= '<a href="#">Visualizza le assenze di tutti gli studenti</a>';
  $menu .= '<a href="#">Inserisci un nuovo studente</a>';
}

// Restituzione del menu
echo $menu;
?>'
```

- Esempio di codice lato client (JavaScript):

```
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Menu</title>
  <script src="<https://code.jquery.com/jquery-3.6.0.min.js>"></script>
</head>
<body>
  <div id="menu"></div>

  <script>
    // Codice per richiedere il menu dal server
    $.ajax({
      url: 'get_menu.php', // Il percorso del file PHP che genera il menu
      method: 'GET',
      success: function(response) {
        $('#menu').html(response); // Aggiunta del menu alla pagina
      },
      error: function() {
        alert('Si è verificato un errore durante il recupero del menu.');
      }
    });
  </script>
</body>
</html>
```

Il codice lato server verifica il ruolo dell'utente loggato e genera un menu personalizzato in base a tale ruolo. Il codice lato client utilizza AJAX per richiedere il menu dal server e lo visualizza nella pagina HTML.

Nota: È importante implementare adeguate misure di sicurezza, come l'autenticazione e l'autorizzazione, per garantire che solo gli utenti autorizzati possano accedere alle funzionalità corrispondenti al loro ruolo.

(Tema 16-17 suppletiva) (con regex)

Un’associazione del terzo settore vuole gestire l’iscrizione on-line di volontari per un’attività di servizio sociale. Il candidato sviluppi, con appropriati linguaggi a scelta sia lato client che lato server, il form on-line per la registrazione dei dati di un volontario, che consenta di memorizzare username, password, cognome, nome, data di nascita, indirizzo di posta elettronica, numero di telefono, posizione lavorativa (lavoratore dipendente, libero professionista, non occupato, pensionato, studente). L’associazione vuole che il form preveda sistemi di gestione delle condizioni di errore, con modalità che il candidato specificherà (es: definizione dei campi obbligatori, l’e-mail deve contenere il carattere ‘@’, la password deve essere composta di un numero minimo di caratteri sia numerici che alfanumerici, il numero di telefono può contenere solo cifre e il carattere ‘/’, …).

Di seguito viene fornito un esempio di form di registrazione per volontari utilizzando HTML, CSS e JavaScript per la validazione dei campi lato client, e PHP per la gestione dei dati lato server.

- HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>Registrazione Volontario</title>
  <style>
    .error {
      color: red;
    }
  </style>
</head>
<body>
  <h1>Registrazione Volontario</h1>
  <form id="registrationForm" method="POST" action="process.php">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username" required>

    <label for="password">Password:</label>
```

```
    <input type="password" id="password" name="password" required>

    <label for="lastname">Cognome:</label>
    <input type="text" id="lastname" name="lastname" required>

    <label for="firstname">Nome:</label>
    <input type="text" id="firstname" name="firstname" required>

    <label for="birthdate">Data di nascita:</label>
    <input type="date" id="birthdate" name="birthdate" required>

    <label for="email">Indirizzo email:</label>
    <input type="email" id="email" name="email" required>

    <label for="phone">Numero di telefono:</label>
    <input type="tel" id="phone" name="phone" required>

    <label for="occupation">Posizione lavorativa:</label>
    <select id="occupation" name="occupation" required>
      <option value="">Seleziona</option>
      <option value="lavoratore_dipendente">Lavoratore dipendente</option>
      <option value="libero_professionista">Libero professionista</option>
      <option value="non_occupato">Non occupato</option>
      <option value="pensionato">Pensionato</option>
      <option value="studente">Studente</option>
    </select>

    <input type="submit" value="Registrati">
  </form>

  <script src="validation.js"></script>
</body>
</html>
```

- JavaScript (validation.js):

```
document.getElementById('registrationForm').addEventListener('submit', function(event) {
  event.preventDefault(); // Previeni l'invio del form

  // Validazione dei campi
  var username = document.getElementById('username').value;
  var password = document.getElementById('password').value;
  var email = document.getElementById('email').value;
  var phone = document.getElementById('phone').value;

  var errors = [];

  if (username.length < 5) {
    errors.push('L\\'username deve essere lungo almeno 5 caratteri.');
  }

  if (password.length < 8 || !/\\d/.test(password) || !/[a-zA-Z]/.test(password)) {
    errors.push('La password deve contenere almeno 8 caratteri e includere sia numeri che lettere.');
  }

  if (!email.includes('@')) {
    errors.push('L\\'indirizzo email non è valido.');
  }

  if (!/^\\d+$/.test(phone) && !/^\\d+\\/\\d+$/.test(phone)) {
    errors.push('Il numero di telefono non è valido.');
  }

  // Mostra gli errori o invia il form
  if (errors.length > 0) {
    var errorElement

 = document.getElementById('errors');
    errorElement.innerHTML = '';
    for (var i = 0; i < errors.length; i++) {
```

```
      errorElement.innerHTML += '<p class="error">' + errors[i] + '</p>';
    }
  } else {
    document.getElementById('registrationForm').submit();
  }
});
```

- PHP (process.php):

```
<?php
// Recupero dei dati inviati dal form
$username = $_POST['username'];
$password = $_POST['password'];
$lastname = $_POST['lastname'];
$firstname = $_POST['firstname'];
$birthdate = $_POST['birthdate'];
$email = $_POST['email'];
$phone = $_POST['phone'];
$occupation = $_POST['occupation'];

// Salvataggio dei dati nel database o elaborazione successiva

// Esempio di risposta per confermare la registrazione
echo 'Registrazione completata con successo!';
?>
```

In questo esempio, il form di registrazione richiede all'utente di inserire i dati richiesti, come username, password, nome, cognome, data di nascita, indirizzo email, numero di telefono e posizione lavorativa. La validazione lato client viene eseguita utilizzando JavaScript, che controlla se i campi soddisfano i requisiti specificati e visualizza eventuali errori. Se tutti i campi sono validi, il form viene inviato al file PHP 'process.php', dove i dati possono essere elaborati ulteriormente o salvati nel database.

È possibile personalizzare ulteriormente la validazione lato client o lato server secondo le specifiche e le esigenze dell'associazione del terzo settore.

(Tema 16-17 suppletiva (algebra relazionale))

Il candidato illustri quali sono gli operatori dell'algebra relazionale discutendone le proprietà anche attraverso l'uso di esempi riferiti al seguente schema relazionale:

- Testo (id, titolo, genere)
- Editore (id, ragione_sociale, città, data_fondazione, logo)
- Pubblica (id_testo, id_edit, anno_pubblic, prezzo)

in cui per la relazione 'Pubblica', i campi 'id_testo' e 'id_edit' referenziano rispettivamente la chiave primaria delle relazioni 'Testo' ed 'Editore'.

L'algebra relazionale è un linguaggio formale per manipolare le relazioni all'interno di un database. Gli operatori dell'algebra relazionale consentono di combinare, selezionare e proiettare i dati all'interno delle relazioni. Di seguito sono elencati i principali operatori dell'algebra relazionale e ne viene data una spiegazione delle proprietà, con esempi basati sullo schema relazionale fornito.

1. Selezione (σ): L'operatore di selezione consente di selezionare le tuple che soddisfano una determinata condizione. Si indica con il simbolo σ seguito dalla condizione di selezione tra parentesi quadre [ ].

   Esempio: Selezionare i testi di genere "Fantascienza".

σ(genere = 'Fantascienza')(Testo)

2. Proiezione (π): L'operatore di proiezione consente di selezionare solo alcune colonne da una relazione. Si indica con il simbolo π seguito dagli attributi da proiettare tra parentesi quadre [ ].

   Esempio: Proiettare solo gli attributi "titolo" e "anno_pubblic" dalla relazione "Pubblica".

   π(titolo, anno_pubblic)(Pubblica)

3. Unione (∪): L'operatore di unione combina le tuple di due relazioni compatibili, eliminando le tuple duplicate. Le relazioni devono avere lo stesso schema.

   Esempio: Unire le relazioni "Pubblica" e "Editore" per ottenere tutte le tuple che rappresentano le pubblicazioni con i rispettivi editori.

   Pubblica ∪ Editore

4. Intersezione (∩): L'operatore di intersezione restituisce le tuple comuni a due relazioni compatibili. Le relazioni devono avere lo stesso schema.

   Esempio: Determinare le tuple che rappresentano i testi pubblicati e che hanno anche un editore corrispondente.

   Pubblica ∩ Editore

5. Differenza (-): L'operatore di differenza restituisce le tuple che sono presenti nella prima relazione ma non nella seconda relazione. Le relazioni devono avere lo stesso schema.

   Esempio: Trovare i testi che sono stati pubblicati ma non hanno un corrispondente editore.

   Testo - Pubblica

6. Prodotto cartesiano (×): L'operatore di prodotto cartesiano combina tutte le tuple di una relazione con tutte le tuple di un'altra relazione. Il risultato ha uno schema che combina gli attributi di entrambe le relazioni.

   Esempio: Ottenere il prodotto cartesiano tra le relazioni "Testo" e "Editore" per ottenere tutte le possibili combinazioni tra testi ed editori.

   Testo × Editore

7. Join (⋈): L'operatore di join combina le tuple di due relazioni in base a una condizione di join specificata. La condizione di join è generalmente basata su un'uguaglianza tra attributi comuni alle due relazioni.

   Esempio: Eseguire un join tra le relazioni "Testo" e "Pubblica" utilizzando l'attributo "id" per ottenere i dettagli dei testi pubblicati.

   Testo ⋈ Pubblica(id)

Gli operatori dell'algebra relazionale offrono un modo potente per interrogare e manipolare i dati all'interno di un database relazionale. Utilizzando questi operatori in combinazione, è possibile ottenere risultati complessi ed estratti specifici di dati a partire dalle relazioni presenti nel database.

(Tema 16-17 sessione straordinaria)

In relazione al tema sviluppato nella prima parte, si consideri che ogni utente registrato ha la possibilità di creare, rivedere, modificare, cancellare le proprie schede di candidatura fino ad una data limite prefissata, dopo di che ne potrà solo prendere visione. Il candidato produca il layout grafico della porzione dell'applicazione Web che consente agli utenti registrati di svolgere le operazioni specificate. Descriva poi la struttura di una guida on-line a supporto degli utenti registrati per illustrare tali funzionalità, scrivendo anche un esempio significativo dei contenuti

Ecco un esempio di layout grafico per la porzione dell'applicazione Web che consente agli utenti registrati di svolgere le operazioni di creazione, revisione, modifica e cancellazione delle proprie schede di candidatura:

```
<h1>Area Personale</h1>

<h2>Benvenuto, [Nome Utente]</h2>

<nav>
  <ul>
    <li><a href="/scheda-creazione">Creazione Scheda</a></li>
    <li><a href="/schede-visualizzazione">Visualizza Schede</a></li>
  </ul>
</nav>

<section>
  <h3>Creazione Scheda di Candidatura</h3>
  <form action="/scheda-creazione" method="POST">
    <!-- Campi per la creazione della scheda -->
    <input type="text" name="titolo" placeholder="Titolo" required>
    <textarea name="descrizione" placeholder="Descrizione" required></textarea>
    <input type="submit" value="Crea Scheda">
  </form>
</section>

<section>
  <h3>Schede di Candidatura</h3>
  <!-- Elenco delle schede create dall'utente -->
  <ul>
    <li>
      <h4>[Titolo Scheda 1]</h4>
      <p>[Descrizione Scheda 1]</p>
      <a href="/scheda-modifica">Modifica</a>
      <a href="/scheda-cancellazione">Cancella</a>
    </li>
    <li>
      <h4>[Titolo Scheda 2]</h4>
      <p>[Descrizione Scheda 2]</p>
      <a href="/scheda-modifica">Modifica</a>
      <a href="/scheda-cancellazione">Cancella</a>
    </li>
    <!-- Altre schede... -->
  </ul>
</section>
```

La struttura di una guida online a supporto degli utenti registrati potrebbe includere i seguenti contenuti:

1. Introduzione: fornire una panoramica generale delle funzionalità offerte nella sezione dell'applicazione Web dedicata agli utenti registrati.
2. Creazione della scheda di candidatura: spiegare come creare una nuova scheda di candidatura, indicando i campi obbligatori e fornendo suggerimenti su come compilare correttamente i campi.

   Esempio di contenuto:

   ```
   <h2>Creazione della scheda di candidatura</h2>
   <p>Per creare una nuova scheda di candidatura, segui questi passaggi:</p>
   <ol>
     <li>Accedi alla tua area personale.</li>
     <li>Fai clic sul link "Creazione Scheda".</li>
     <li>Compila i campi richiesti, come il titolo e la descrizione della scheda.</li>
     <li>Fai clic sul pulsante "Crea Scheda" per salvare la tua candidatura.</li>
   </ol>
   <p>Ricorda di fornire tutte le informazioni richieste in modo accurato e completo per massimizzare le possibilità di su
   ccesso della tua candidatura.</p>
   ```

3. Visualizzazione delle schede di candidatura: spiegare come visualizzare le schede di candidatura create dall'utente, fornendo informazioni su come leggere i dettagli delle schede e prendere visione delle candidature.

   Esempio di contenuto:

```
<h2>Visualizzazione delle schede di candidatura</h2>
<p>Per visualizzare le tue schede di candidatura, segui questi passaggi:</p>
<ol>
  <li>Accedi alla tua area personale.</li>
  <li>Fai clic sul link "Visualizza Schede".</li>
  <li>Ti verrà mostrato un elenco delle schede di candidatura che hai creato.</li>
  <li>Fai clic sul titolo di una scheda per visualizzare i dettagli e prendere visione della candidatura.</li>
</ol>
<p>Puoi anche trovare altre informazioni, come la data di creazione della scheda e il prezzo associato, nella visualizz
azione delle schede.</p>
```

4. Modifica e cancellazione delle schede di candidatura: spiegare come effettuare modifiche alle schede di candidatura esistenti o cancellarle, tenendo conto delle restrizioni sulla data limite per le operazioni.

   Esempio di contenuto:

```
<h2>Modifica e cancellazione delle schede di candidatura</h2>
<p>Puoi modificare o cancellare le tue schede di candidatura fino alla data limite prefissata. Ecco come:</p>
<ol>
  <li>Accedi alla tua area personale.</li>
  <li>Fai clic sul titolo di una scheda di candidatura che desideri modificare o cancellare.</li>
  <li>Se la data limite non è ancora scaduta, vedrai i link "Modifica" e "Cancella".</li>
  <li>Fai clic sul link "Modifica" per aprire la scheda in modalità di modifica.</li>
  <li>Apporta le modifiche necessarie e fai clic sul pulsante "Salva" per confermare le modifiche.</li>
  <li>Se desideri cancellare la scheda, fai clic sul link "Cancella" e conferma l'operazione.</li>
</ol>
<p>Ricorda che dopo la data limite, potrai solo prendere visione delle schede di candidatura senza la possibilità di ap
portare modifiche o cancellarle.</p>
```

La guida online fornirebbe ulteriori dettagli e istruzioni specifiche per ogni operazione disponibile agli utenti registrati, guidandoli passo dopo passo attraverso il processo di gestione delle schede di candidatura.

(Tema 16-17 sessione straordinaria)

Il candidato presenti una panoramica delle tecnologie di sua conoscenza per lo sviluppo di applicazioni Web dinamiche. Approfondisca le differenze tra tali tecnologie, nelle funzionalità ed utilizzi tipici, e le esemplifichi, facendo riferimento al seguente caso concreto: in una applicazione Web viene richiesto di compilare tre campi di residenza con Regione, Provincia e Comune, che vengono impostati tramite tre listbox in cui la seconda e la terza presentano valori diversificati in funzione di quanto selezionato rispettivamente nella prima e nella seconda (es: se come regione viene selezionata Liguria dall'elenco delle Regioni, la listbox Provincia dovrà mostrare i valori Imperia, Savona, Genova, La Spezia).

Per lo sviluppo di applicazioni Web dinamiche, ci sono diverse tecnologie tra cui scegliere, ognuna con le proprie funzionalità e utilizzi tipici. Alcune delle principali tecnologie per lo sviluppo di applicazioni Web dinamiche includono:

1. HTML (HyperText Markup Language): è il linguaggio di markup standard per la creazione di pagine Web. Viene utilizzato per definire la struttura e il contenuto di una pagina Web.
2. CSS (Cascading Style Sheets): è un linguaggio per la formattazione delle pagine Web. Viene utilizzato per definire l'aspetto visivo di una pagina Web, come il layout, i colori, i tipi di carattere, ecc.

3. JavaScript: è un linguaggio di scripting ampiamente utilizzato per aggiungere interattività e funzionalità dinamiche alle pagine Web. JavaScript può essere utilizzato per manipolare il contenuto della pagina, rispondere agli eventi dell'utente e comunicare con il server.
4. AJAX (Asynchronous JavaScript and XML): è una tecnica di sviluppo Web che consente di inviare e ricevere dati dal server in background senza dover ricaricare l'intera pagina. AJAX sfrutta JavaScript e XML (o altri formati dati come JSON) per effettuare chiamate asincrone al server.
5. PHP (Hypertext Preprocessor): è un linguaggio di scripting lato server ampiamente utilizzato per lo sviluppo Web. PHP viene eseguito sul server e può generare dinamicamente pagine HTML.
6. ASP.NET: è un framework di sviluppo Web sviluppato da Microsoft. Utilizza diversi linguaggi di programmazione, come C# o Visual Basic, per creare applicazioni Web dinamiche. ASP.NET viene eseguito sul server e supporta diversi modelli di programmazione, come Web Forms e MVC (Model-View-Controller).
7. Ruby on Rails: è un framework di sviluppo Web scritto in Ruby. Ruby on Rails favorisce la scrittura di codice pulito e conciso, seguendo il principio di "convenzione sopra configurazione". È noto per la sua facilità di sviluppo e per il supporto di molte funzionalità Web comuni.

Nel caso specifico della compilazione dei campi di residenza (Regione, Provincia e Comune) con valori dinamici basati sulle selezioni precedenti, è possibile utilizzare una combinazione di JavaScript e AJAX per ottenere questo risultato. Ecco un esempio di come si potrebbe implementare:

1. Definizione delle tre listbox nel codice HTML:

```
<select id="regione">
  <option value="">Seleziona Regione</option>
  <option value="liguria">Liguria</option>
  <!-- Altre regioni... -->
</select>

<select id="provincia">
  <option value="">Seleziona Provincia</option>
</select>

<select id="comune">
  <option value="">Seleziona Comune</option>
</select>
```

2. Utilizzo di JavaScript per gestire gli eventi di selezione delle listbox e aggiornare dinamicamente i valori delle listbox successive:

```
// Ottenere le referenze alle listbox
var regioneSelect = document.getElementById('regione');
var provinciaSelect = document

.getElementById('provincia');
var comuneSelect = document.getElementById('comune');

// Definire un oggetto che mappa le relazioni tra regioni, province e comuni
var regioneProvinciaComuni = {
  'liguria': {
    'imperia': ['Comune1', 'Comune2', 'Comune3'],
    'savona': ['Comune4', 'Comune5', 'Comune6'],
    'genova': ['Comune7', 'Comune8', 'Comune9'],
    'la_spezia': ['Comune10', 'Comune11', 'Comune12']
  },
  // Altre regioni...
};

// Aggiungere un event listener per l'evento di cambio selezione sulla listbox delle regioni
regioneSelect.addEventListener('change', function() {
```

```
  // Ottenere il valore selezionato della regione
  var regione = regioneSelect.value;

  // Pulire le listbox delle province e dei comuni
  provinciaSelect.innerHTML = '<option value="">Seleziona Provincia</option>';
  comuneSelect.innerHTML = '<option value="">Seleziona Comune</option>';

  // Selezionare le province basate sulla regione selezionata
  if (regioneProvinciaComuni.hasOwnProperty(regione)) {
    var province = Object.keys(regioneProvinciaComuni[regione]);

    // Aggiungere le province alla listbox delle province
    province.forEach(function(provincia) {
      var option = document.createElement('option');
      option.value = provincia;
      option.text = provincia;
      provinciaSelect.appendChild(option);
    });
  }
});

// Aggiungere un event listener per l'evento di cambio selezione sulla listbox delle province
provinciaSelect.addEventListener('change', function() {
  // Ottenere il valore selezionato della regione e della provincia
  var regione = regioneSelect.value;
  var provincia = provinciaSelect.value;

  // Pulire la listbox dei comuni
  comuneSelect.innerHTML = '<option value="">Seleziona Comune</option>';

  // Selezionare i comuni basati sulla regione e provincia selezionati
  if (regioneProvinciaComuni.hasOwnProperty(regione) && regioneProvinciaComuni[regione].hasOwnProperty(provincia)) {
    var comuni = regioneProvinciaComuni[regione][provincia];

    // Aggiungere i comuni alla listbox dei comuni
    comuni.forEach(function(comune) {
      var option = document.createElement('option');
      option.value = comune;
      option.text = comune;
      comuneSelect.appendChild(option);
    });
  }
});
```

In questo esempio, quando l'utente seleziona una regione, la listbox delle province viene popolata con le province corrispondenti. Successivamente, quando viene selezionata una provincia, la listbox dei comuni viene popolata con i comuni corrispondenti alla regione e alla provincia selezionate.

È possibile personalizzare l'esempio aggiungendo ulteriori valori per le regioni, province e comuni, o modificando la struttura dei dati in base alle specifiche necessità del caso concreto.

(Tema 16-17 sessione straordinaria)

Il candidato illustri le attività tipiche di un amministratore di una base di dati, corredandole con esempi significativi di comandi SQL specifici per lo svolgimento delle funzioni a lui riservate.

L'amministratore di una base di dati svolge diverse attività per garantire il corretto funzionamento e la gestione efficiente del sistema. Alcune delle attività tipiche di un amministratore di database includono:

1. Installazione e configurazione del database: L'amministratore è responsabile dell'installazione del sistema di gestione del database (DBMS) e della sua configurazione iniziale. Questo può includere la scelta delle impostazioni di configurazione, la creazione dei database e la definizione di parametri di sistema.
    - Esempio di comando SQL: Creazione di un nuovo database in MySQL:

```
CREATE DATABASE nome_database;
```

2. Gestione degli utenti e delle autorizzazioni: L'amministratore assegna e gestisce gli account degli utenti nel database, definendo i privilegi e le autorizzazioni appropriati. Questo include la creazione di nuovi utenti, la modifica delle autorizzazioni esistenti e la revoca dei privilegi quando necessario.
   - Esempio di comando SQL: Creazione di un nuovo utente con privilegi di lettura/scrittura su una tabella in PostgreSQL:

```
CREATE USER nome_utente WITH PASSWORD 'password';
GRANT SELECT, INSERT, UPDATE, DELETE ON nome_tabella TO nome_utente;
```

3. Monitoraggio delle prestazioni: L'amministratore monitora le prestazioni del database per garantire che il sistema funzioni in modo efficiente. Questo può includere l'analisi delle query lente, il monitoraggio dell'utilizzo delle risorse e l'ottimizzazione delle tabelle e degli indici per migliorare le prestazioni.
   - Esempio di comando SQL: Esecuzione di una spiegazione dell'esecuzione di una query in Oracle per identificare possibili ottimizzazioni:

```
EXPLAIN PLAN FOR SELECT * FROM nome_tabella;
```

4. Backup e ripristino dei dati: L'amministratore pianifica e gestisce i processi di backup e ripristino per garantire la disponibilità e l'integrità dei dati. Questo include la creazione di copie di backup regolari dei database e la pianificazione di procedure di ripristino in caso di perdita di dati o guasti del sistema.
   - Esempio di comando SQL: Esecuzione di un backup di un database MySQL:

```
mysqldump -u nome_utente -p nome_database > nome_file.sql;
```

5. Monitoraggio della sicurezza: L'amministratore gestisce la sicurezza dei dati nel database, implementando meccanismi di autenticazione e autorizzazione per proteggere l'accesso ai dati sensibili. Monitora anche l'attività degli utenti per individuare eventuali anomalie o attività sospette.
   - Esempio di comando SQL: Visualizzazione dei privilegi di un utente in SQL Server:

```
EXEC sp_helprotect 'nome_utente';
```

Questi sono solo alcuni esempi delle attività tipiche di un amministratore di database, e i comandi SQL possono variare a seconda del DBMS utilizzato. L'amministratore deve avere una buona conoscenza delle funzionalità del DBMS e delle migliori pratiche di gestione del database per garantire un'efficace amministrazione del sistema.

(Tema 14-15 ordinario)

Si consideri la seguente tabella:

Cognome	Nome	Telefono	Livello	Tutor	Tel-tutor	Anticipo versato
Verdi	Luisa	345698741	avanzato	Bianca	334563215	100
Neri	Enrico	348523698	avanzato	Carlo	369852147	150
Rosi	Rosa	347532159	base	Alessio	333214569	120
Bianchi	Paolo	341236547	base	Carlo	369852147	150
Rossi	Mario	349567890	base	Carlo	369852147	90
Neri	Enrico	348523698	complementi	Dina	373564987	100

Il candidato verifichi le proprietà di normalizzazione e proponga uno schema equivalente che rispetti la 3^ Forma Normale, motivando le scelte effettuate.

~ I database relazionali sono spesso progettati in modo da rispettare le normalizzazioni, che sono una serie di regole per organizzare le tabelle in modo efficiente ed evitare ridondanze e anomalie dei dati. Ci sono tre forme normali (1NF, 2NF e 3NF) che vengono comunemente utilizzate. Ecco una spiegazione dettagliata di ciascuna forma normale:

1. Prima Forma Normale (1NF):
   - Ogni attributo di una tabella deve contenere solo valori atomici, cioè non può essere suddiviso ulteriormente. Questo significa che ogni attributo deve contenere un solo valore e non può essere una lista o un insieme di valori.
   - Ogni riga in una tabella deve essere unica, quindi non possono esserci duplicati.
2. Seconda Forma Normale (2NF):
   - Una tabella è nella 2NF se soddisfa la 1NF e se tutti gli attributi non chiave dipendono completamente dalla chiave primaria.
   - Questo significa che ogni attributo non chiave deve dipendere da tutta la chiave primaria, non solo da una parte di essa.
   - Se un attributo non chiave dipende solo da una parte della chiave primaria, allora dovrebbe essere spostato in una tabella separata insieme a quella parte della chiave primaria.
3. Terza Forma Normale (3NF):
   - Una tabella è nella 3NF se soddisfa la 2NF e se tutti gli attributi non chiave dipendono solo dalla chiave primaria e non da altri attributi non chiave.
   - Questo significa che non ci dovrebbero essere dipendenze transitie o dipendenze funzionali tra gli attributi non chiave.
   - Se un attributo non chiave dipende da un altro attributo non chiave, allora dovrebbe essere spostato in una tabella separata.

La terza forma normale (3NF) è considerata la forma normale più comune nella progettazione di database relazionali. Assicura che le tabelle siano strutturate in modo da minimizzare la ridondanza dei dati e le anomalie di aggiornamento.

Per applicare la terza forma normale allo schema fornito precedentemente, dovremmo esaminare le dipendenze funzionali tra gli attributi delle tabelle e suddividerle in tabelle separate in base alle dipendenze. Tuttavia, dallo schema dato non sono chiare le dipendenze funzionali tra gli attributi, quindi non è possibile proporre uno schema equivalente che rispetti la 3NF senza ulteriori informazioni.

In generale, l'obiettivo principale della normalizzazione è ridurre la ridondanza dei dati e garantire che le tabelle siano organizzate in modo coerente, consentendo una gestione più efficiente dei dati e mantenendo l'integrità e la coerenza dei dati stessi.

(Tema 14-15 ordinario)

Nella formalizzazione di uno schema concettuale, le associazioni tra entità sono caratterizzate da una cardinalità: esponga il significato e la casistica che si può presentare.

Nella formalizzazione di uno schema concettuale, le associazioni tra entità sono caratterizzate dalla cardinalità, che definisce il numero di occorrenze che possono essere associate tra due entità correlate. La cardinalità descrive la natura e la quantità delle relazioni tra le entità coinvolte. Esistono diverse casistiche di cardinalità che si possono presentare:

1. Cardinalità Uno a Uno (1:1):
   - In questa casistica, un'istanza di un'entità è associata a un'unica istanza dell'altra entità, e viceversa.
   - Ad esempio, consideriamo le entità "Persona" e "Passaporto". Ogni persona può avere un solo passaporto e ogni passaporto è associato a una sola persona.
2. Cardinalità Uno a Molti (1:N):
   - In questa casistica, un'istanza di un'entità è associata a molte istanze dell'altra entità, ma ogni istanza dell'altra entità è associata a un'unica istanza dell'entità.
   - Ad esempio, consideriamo le entità "Dipartimento" e "Impiegato". Un dipartimento può avere molti impiegati, ma ogni impiegato è assegnato a un solo dipartimento.
3. Cardinalità Molti a Molti (N:M):
   - In questa casistica, molte istanze di un'entità sono associate a molte istanze dell'altra entità, e viceversa.
   - Per rappresentare questa casistica nel modello concettuale, viene introdotta una tabella di associazione (tabella di giunzione) che collega le due entità.
   - Ad esempio, consideriamo le entità "Studente" e "Corso". Uno studente può iscriversi a molti corsi e un corso può avere molti studenti. La tabella di associazione conterrà le chiavi primarie delle entità coinvolte e ulteriori attributi specifici dell'associazione stessa, come ad esempio la data di iscrizione.

È importante definire correttamente la cardinalità tra le entità nel modello concettuale in quanto influisce sulla progettazione dello schema logico e fisico del database. La scelta della cardinalità corretta dipende dalla semantica del dominio applicativo e dalle specifiche dei requisiti del sistema.

(Tema 14-15 simulazione)

In relazione al tema proposto nella prima parte, indichi come intende affrontare la gestione degli accessi riservati agli operatori dei caseifici per lo svolgimento delle loro funzioni.

~ Per affrontare la gestione degli accessi riservati agli operatori dei caseifici, è necessario implementare un sistema di autenticazione e autorizzazione sicuro. Ecco alcuni passaggi che possono essere seguiti per gestire gli accessi riservati:

1. Creazione di account utente: Ogni operatore dei caseifici deve avere un account utente unico per accedere all'applicazione. Durante la registrazione, verranno richieste informazioni come nome, cognome, indirizzo email e password.
2. Autenticazione: Per verificare l'identità degli operatori, sarà necessario implementare un meccanismo di autenticazione sicuro. Questo può essere fatto utilizzando una combinazione di username e password, o utilizzando tecnologie più avanzate come l'autenticazione a due fattori o l'autenticazione basata su certificati digitali.

3. Autorizzazione: Una volta autenticati, gli operatori devono avere accesso solo alle funzioni e ai dati che sono pertinenti alle loro responsabilità. È importante definire i ruoli e i livelli di autorizzazione in base alle funzioni svolte dagli operatori. Ad esempio, ci potrebbero essere ruoli come "Operatore di produzione", "Supervisore", "Gestore dell'inventario", ognuno con autorizzazioni specifiche.
4. Controllo degli accessi: È importante implementare un sistema di controllo degli accessi per garantire che gli operatori possano accedere solo alle risorse e ai dati appropriati. Ciò può essere fatto tramite meccanismi come la definizione di regole di accesso basate sui ruoli degli utenti o l'assegnazione di autorizzazioni specifiche a livello di database o di applicazione.
5. Monitoraggio degli accessi: È consigliabile tenere traccia degli accessi degli operatori ai fini di audit e sicurezza. Registrare gli accessi consentirà di identificare eventuali attività sospette o anomalie nel sistema.
6. Protezione dei dati sensibili: I caseifici possono gestire dati sensibili come informazioni personali degli operatori, informazioni di produzione e dati finanziari. È fondamentale implementare misure di sicurezza per proteggere questi dati, come crittografia, protezione dei dati in transito e inattività automatica della sessione.

È importante sottolineare che la gestione degli accessi riservati richiede una progettazione attenta e una robusta implementazione delle misure di sicurezza. È consigliabile coinvolgere esperti in sicurezza informatica per garantire che vengano seguite le migliori pratiche e che il sistema sia protetto da potenziali minacce o violazioni della sicurezza.

(Tema 14-15 simulazione)

In relazione al tema proposto nella prima parte, sviluppi la query SQL per calcolare la percentuale di forme di seconda scelta prodotte annualmente da un certo caseificio (sul totale delle forme da lui prodotte annualmente).

~ Per calcolare la percentuale di forme di seconda scelta prodotte annualmente da un caseificio rispetto al totale delle forme prodotte, puoi utilizzare la seguente query SQL:

```
SELECT (COUNT(*) * 100.0 / (SELECT COUNT(*) FROM Forme WHERE anno_produzione = <anno_specifico>)) AS percentuale
FROM Forme
WHERE anno_produzione = <anno_specifico> AND tipo = 'Seconda scelta';
```

In questa query, dovrai sostituire `<anno_specifico>` con l'anno per cui desideri calcolare la percentuale di forme di seconda scelta prodotte. La tabella "Forme" rappresenta la tabella che contiene le informazioni sulle forme prodotte, con i campi "anno_produzione" che indica l'anno di produzione e "tipo" che indica il tipo di forma (prima scelta, seconda scelta, etc.).

La query utilizza una subquery per calcolare il totale delle forme prodotte nell'anno specificato. Successivamente, viene eseguita la query principale che conta il numero di forme di seconda scelta prodotte nell'anno specificato e calcola la percentuale dividendo il numero di forme di seconda scelta per il totale delle forme prodotte e moltiplicando per 100.0 per ottenere la percentuale corretta.

Il risultato della query sarà la percentuale di forme di seconda scelta prodotte annualmente dal caseificio rispetto al totale delle forme prodotte nell'anno specificato.

(Tema 14-15 simulazione)

Illustri, anche servendosi di esempi, il concetto di “vista” in una base di dati.

In una base di dati, una vista è una rappresentazione virtuale di una porzione specifica dei dati presenti nelle tabelle. Essa può essere considerata come una "tabella virtuale" che viene creata utilizzando una query SQL e che restituisce

un insieme di righe e colonne basate sui dati presenti nelle tabelle sottostanti. Le viste forniscono un'interfaccia semplificata e personalizzata per gli utenti, consentendo loro di accedere solo alle informazioni rilevanti e di eseguire operazioni specifiche senza dover accedere direttamente alle tabelle originali.

Ecco un esempio per comprendere meglio il concetto di vista:

Supponiamo di avere due tabelle: "Clienti" e "Ordini". La tabella "Clienti" contiene informazioni sui clienti come nome, cognome, indirizzo, etc., mentre la tabella "Ordini" contiene informazioni sugli ordini effettuati dai clienti come numero ordine, data ordine, importo, etc. Ogni cliente può avere uno o più ordini associati.

Possiamo creare una vista chiamata "ClientiConOrdini" per ottenere una visualizzazione dei clienti insieme ai loro ordini corrispondenti. La vista può essere creata utilizzando una query che unisce le tabelle "Clienti" e "Ordini" basandosi su una colonna comune come l'ID cliente.

Esempio di creazione di una vista in SQL:

```
CREATE VIEW ClientiConOrdini AS
SELECT c.Nome, c.Cognome, o.NumeroOrdine, o.DataOrdine, o.Importo
FROM Clienti c
JOIN Ordini o ON c.IDCliente = o.IDCliente;
```

Una volta creata la vista, possiamo utilizzarla come una tabella regolare nelle query. Ad esempio, possiamo eseguire una semplice query sulla vista per ottenere l'elenco dei clienti insieme ai loro ordini:

```
SELECT *
FROM ClientiConOrdini;
```

La vista "ClientiConOrdini" restituirà un insieme di righe contenenti i dati dei clienti insieme ai dati degli ordini corrispondenti.

Le viste offrono diversi vantaggi, tra cui:

1. Semplificazione dei dati: Le viste permettono di creare una visione personalizzata dei dati, includendo solo le colonne necessarie e combinando le informazioni da diverse tabelle, rendendo più semplice l'accesso e l'analisi dei dati.
2. Sicurezza: Le viste possono essere utilizzate per nascondere determinate colonne o righe sensibili, consentendo agli utenti di accedere solo alle informazioni rilevanti per le loro operazioni, mantenendo allo stesso tempo la sicurezza dei dati.
3. Astrazione dei dati: Le viste forniscono un'astrazione dei dati sottostanti, consentendo di apportare modifiche alla struttura delle tabelle senza influenzare direttamente le query che utilizzano la vista. Ciò facilita la gestione dei cambiamenti nella struttura del database senza dover modificare le applicazioni che dipendono da esse.
4. Prestazioni: Le viste possono anche essere utilizzate per ottimizzare le prestazioni delle query. Ad esempio, una vista può essere cre

ata per aggregare dati complessi o eseguire operazioni di join complesse, consentendo alle query successive di accedere ai dati già elaborati e migliorando le prestazioni complessive.

In sintesi, le viste forniscono una visione personalizzata e semplificata dei dati in una base di dati, consentendo agli utenti di accedere solo alle informazioni pertinenti per le loro esigenze e offrendo vantaggi in termini di sicurezza, flessibilità e prestazioni delle query.

(Tema 14-15 simulazione)

Descriva le tipologie di linguaggi utilizzate in un DBMS.

In un Database Management System (DBMS), vengono utilizzati diversi tipi di linguaggi per interagire con il database e gestire le operazioni di manipolazione, definizione e controllo dei dati. Ecco una panoramica delle principali tipologie di linguaggi utilizzate in un DBMS:

1. SQL (Structured Query Language): SQL è il linguaggio standard per la gestione di database relazionali. Esso consente di eseguire operazioni di interrogazione (SELECT), inserimento (INSERT), aggiornamento (UPDATE) ed eliminazione (DELETE) dei dati nelle tabelle del database. SQL permette anche di definire la struttura del database (DDL - Data Definition Language) mediante comandi come CREATE, ALTER e DROP per la creazione, modifica e cancellazione di tabelle, indici, viste e altri oggetti del database.
2. DML (Data Manipulation Language): DML è un sottinsieme di SQL e include i comandi per la manipolazione dei dati. Oltre alle operazioni di interrogazione, inserimento, aggiornamento ed eliminazione, DML può includere anche operazioni di controllo dei dati come la validazione e la gestione delle transazioni. In SQL, i comandi DML principali sono SELECT, INSERT, UPDATE, DELETE.
3. DDL (Data Definition Language): DDL è un sottinsieme di SQL e include i comandi per la definizione della struttura del database e degli oggetti in esso contenuti. I comandi DDL consentono di creare, modificare e cancellare tabelle, viste, indici, vincoli di integrità, procedure, trigger e altre entità del database. Alcuni esempi di comandi DDL sono CREATE, ALTER, DROP.
4. DCL (Data Control Language): DCL è un sottinsieme di SQL e include i comandi per il controllo degli accessi e delle autorizzazioni sui dati nel database. I comandi DCL consentono di concedere o revocare privilegi agli utenti o ai ruoli nel database. Alcuni esempi di comandi DCL sono GRANT, REVOKE.
5. TCL (Transaction Control Language): TCL è un sottinsieme di SQL e include i comandi per il controllo delle transazioni nel database. I comandi TCL consentono di definire e gestire le transazioni, ovvero insiemi di operazioni che devono essere eseguite come un'unità atomica e che devono rispettare le proprietà di ACID (Atomicity, Consistency, Isolation, Durability). Alcuni esempi di comandi TCL sono COMMIT, ROLLBACK, SAVEPOINT.

Oltre a questi linguaggi specifici per la gestione dei dati, ci sono anche linguaggi di programmazione generali come Java, Python, C# che possono essere utilizzati per sviluppare applicazioni che interagiscono con il database attraverso API o driver specifici. Questi linguaggi consentono di eseguire operazioni più complesse, logica di business personalizzata, elaborazioni dei dati e interazioni con altri sistemi esterni.

(Tema 14-15 suppletiva)

In relazione al tema proposto nella prima parte, in riferimento alle funzioni elencate ai punti A, B, C, D, indicare le strategie di massima da adottare per consentire l'accesso diversificato da parte di categorie di utenti secondo lo schema seguente: - funzione A) per uno o più amministratori di sistema; - funzione B) per i gestori locali, limitatamente al proprio centro; - funzione C) per utenti registrati; - funzione D) per utenti anonimi.

~ Per consentire l'accesso diversificato da parte di categorie di utenti secondo lo schema descritto, è possibile adottare le seguenti strategie:

1. Autenticazione e autorizzazione: Implementare un sistema di autenticazione che richieda alle persone di fornire credenziali valide per accedere all'applicazione. Le categorie di utenti dovrebbero essere associate a ruoli o permessi specifici. Ad esempio, gli amministratori di sistema avranno privilegi estesi (funzione A), i gestori locali avranno accesso limitato al proprio centro (funzione B), gli utenti registrati avranno accesso alle funzioni specifiche

per gli utenti (funzione C), mentre gli utenti anonimi potrebbero avere solo accesso a funzionalità limitate (funzione D).

2. Controllo degli accessi: Implementare un sistema di controllo degli accessi che limiti l'accesso alle funzionalità e ai dati in base ai ruoli o ai permessi degli utenti. Questo può essere realizzato mediante l'assegnazione di autorizzazioni specifiche a ciascuna funzione o risorsa dell'applicazione. Ad esempio, gli amministratori di sistema avranno accesso completo a tutte le funzioni e i dati (funzione A), mentre i gestori locali avranno accesso solo alle funzioni e ai dati relativi al proprio centro (funzione B), e così via.
3. Segmentazione dei dati: Se necessario, è possibile suddividere i dati in base alle categorie di utenti per garantire che ciascuna categoria possa accedere solo ai dati pertinenti. Ad esempio, i gestori locali potrebbero avere accesso solo ai dati relativi al proprio centro, mentre gli utenti registrati potrebbero accedere solo ai propri dati personali.
4. Interfaccia utente personalizzata: Creare un'interfaccia utente che mostri solo le funzioni e le opzioni pertinenti per ciascuna categoria di utenti. In questo modo, gli utenti vedranno solo ciò che è rilevante per il loro ruolo o livello di accesso, semplificando l'uso dell'applicazione e riducendo la confusione.
5. Registrazione delle attività: Tenere traccia delle attività degli utenti, registrando le azioni eseguite e i dati a cui hanno accesso. Questo può essere utile per scopi di audit e sicurezza, consentendo di monitorare le operazioni svolte dagli utenti e individuare eventuali violazioni o utilizzi impropri dell'applicazione.
6. Mantenere la sicurezza: Implementare meccanismi di sicurezza appropriati come l'uso di crittografia per la trasmissione dei dati sensibili, controlli per prevenire attacchi di tipo SQL injection o Cross-Site Scripting (XSS), e regole di gestione delle password per garantire che le credenziali degli utenti siano sufficientemente sicure.

È importante notare che le strategie sopra descritte sono generali e potrebbero richiedere personalizzazioni in base alle specifiche esigenze dell'applicazione e dell'ambiente in cui viene utilizzata. La corretta implementazione di queste strategie richiede una valutazione approfondita dei requisiti

(Tema 14-15 suppletiva)

Spieghi la differenza tra linguaggi di sviluppo web lato server e lato client, evidenziandone le specificità e i campi di utilizzo.

I linguaggi di sviluppo web lato server e lato client sono utilizzati per gestire diverse parti di un'applicazione web e hanno specificità e campi di utilizzo distinti. Di seguito, spiegherò la differenza tra i due tipi di linguaggi:

Linguaggi di sviluppo web lato server:
I linguaggi di sviluppo web lato server sono utilizzati per gestire la logica e il funzionamento del server che ospita l'applicazione web. Questi linguaggi vengono eseguiti sul server e generano dinamicamente il codice HTML, CSS e JavaScript che viene inviato al browser del client per essere visualizzato. Alcuni esempi di linguaggi di sviluppo web lato server sono:

1. PHP: È un linguaggio di scripting ampiamente utilizzato per lo sviluppo web. Può essere incorporato direttamente nel codice HTML e consente di generare pagine web dinamiche.
2. Python: È un linguaggio di programmazione versatile che può essere utilizzato per lo sviluppo web. Esistono diversi framework web in Python, come Django e Flask, che semplificano lo sviluppo di applicazioni web complesse.
3. Java: È un linguaggio di programmazione popolare utilizzato per sviluppare applicazioni web enterprise. Framework come Java Servlets e JavaServer Pages (JSP) sono utilizzati per gestire le richieste dei client e generare pagine web dinamiche.

I linguaggi di sviluppo web lato server sono responsabili di elaborare le richieste del client, interagire con il database, gestire la logica di business, autenticare gli utenti e generare il codice HTML da inviare al browser del client. Questi linguaggi sono adatti per implementare funzionalità complesse e gestire dati sensibili.

Linguaggi di sviluppo web lato client:
I linguaggi di sviluppo web lato client sono utilizzati per gestire l'interfaccia utente e l'interazione con l'utente direttamente nel browser. Questi linguaggi vengono eseguiti sul computer o dispositivo del cliente e sono responsabili di rendere dinamiche e interattive le pagine web. Alcuni esempi di linguaggi di sviluppo web lato client sono:

1. HTML (Hypertext Markup Language): È il linguaggio di marcatura utilizzato per definire la struttura e il contenuto delle pagine web.
2. CSS (Cascading Style Sheets): È utilizzato per definire l'aspetto e lo stile delle pagine web, come colori, font, layout e animazioni.
3. JavaScript: È un linguaggio di scripting che viene eseguito direttamente nel browser del client. Viene utilizzato per aggiungere interattività, validazione dei dati, manipolazione del DOM (Document Object Model) e comunicazione con il server mediante richieste asincrone (AJAX).

I linguaggi di sviluppo web lato client sono responsabili di rendere le pagine web interattive, gestire eventi come clic del mouse e pressione dei tasti, effettuare convalida dei dati lato client e interagire con il server per ottenere o inviare dati in modo asincrono.

In sintesi, i linguaggi di sviluppo web lato server sono utilizzati per gestire la logica e l'elaborazione sul server, mentre i linguaggi di sviluppo web lato client sono utilizzati per gestire l'interfaccia utente e l'interazione diretta con l'utente nel browser. Entrambi i tipi di linguaggi sono essenziali per lo sviluppo di applicazioni web complete e interattive.

(Tema 14-15 suppletiva)

Illustri, anche attraverso un esempio, le proprietà di una relazione in Seconda Forma Normale ed i tipi di anomalie che possono presentarsi se una relazione non la rispetta.

La Seconda Forma Normale (2NF) è una proprietà che una relazione deve soddisfare per garantire una buona struttura dei dati e ridurre le anomalie durante la gestione dei dati. Per soddisfare la 2NF, una relazione deve prima essere nella Prima Forma Normale (1NF) e inoltre non deve avere dipendenze parziali.

La dipendenza parziale si verifica quando gli attributi di una relazione dipendono solo da una parte della chiave primaria invece che dall'intera chiave primaria. Ciò può portare a una ridondanza dei dati e può causare anomalie durante le operazioni di inserimento, aggiornamento e cancellazione dei dati.

Per illustrare le proprietà della 2NF e le anomalie associate, consideriamo lo schema relazionale seguente:

Tabella Studenti

- Matricola (Chiave primaria)
- Nome
- Cognome
- CodiceCorso
- NomeCorso

Nella tabella degli studenti, abbiamo la seguente dipendenza funzionale:

Matricola -> Nome, Cognome, CodiceCorso, NomeCorso

In questo caso, la chiave primaria è "Matricola". Tuttavia, i campi "Nome", "Cognome", "CodiceCorso" e "NomeCorso" dipendono solo dalla "Matricola" e non dall'intera chiave primaria. Questo crea una dipendenza parziale che viola la 2NF.

Le anomalie che possono verificarsi a causa di questa violazione includono:

1. Anomalia di inserimento: Se si desidera inserire un nuovo studente ma non si conosce ancora il corso, sarebbe necessario inserire un valore nullo o temporaneo per il campo "CodiceCorso" e "NomeCorso", creando dati incompleti o inconsistenti.
2. Anomalia di aggiornamento: Se si desidera modificare il nome di un corso, sarà necessario aggiornare tutte le righe degli studenti che sono associate a quel corso, anche se il corso non ha subito alcuna modifica.
3. Anomalia di cancellazione: Se si cancella un corso, si rischia di cancellare anche tutte le righe degli studenti associati a quel corso, anche se gli studenti stessi non hanno subito alcuna modifica.

Per risolvere queste anomalie e soddisfare la 2NF, si può separare la tabella degli studenti in due tabelle separate: una tabella "Studenti" e una tabella "Corsi". In questo modo, la dipendenza funzionale sarà soddisfatta e non ci saranno dipendenze parziali.

Tabella Studenti

- Matricola (Chiave primaria)
- Nome
- Cognome
- CodiceCorso (Chiave esterna)

Tabella Corsi

- CodiceCorso (Chiave primaria)
- NomeCorso

Ora, gli attributi "CodiceCorso" e "NomeCorso" dipenderanno interamente dalla chiave primaria "CodiceCorso" nella tabella "Corsi". Ciò elimina le anomalie e migliora l'organizzazione dei dati.

In sintesi, la Seconda Forma Normale (2NF) garantisce una struttura dei dati più coerente e riduce le anomalie di inserimento, aggiornamento e cancellazione dei dati.

Temi possibili mai usciti

- trigger
- stored procedure
- database NoSQL

1. Trigger: Un trigger è un oggetto di un database che viene attivato automaticamente quando si verifica un evento specifico, come l'inserimento, l'aggiornamento o la cancellazione di dati in una tabella. Un trigger può essere configurato per eseguire un blocco di codice o un'azione specifica in risposta all'evento. Ad esempio, si potrebbe definire un trigger per calcolare automaticamente il totale di una fattura ogni volta che viene inserita una nuova riga nella tabella degli articoli.
2. Stored Procedure: Una stored procedure è una raccolta di istruzioni SQL predefinite che viene memorizzata nel database e può essere richiamata e eseguita successivamente. Una stored procedure può accettare parametri in input e restituire risultati come output. Viene utilizzata per eseguire operazioni complesse o ripetitive sul database

in modo efficiente e sicuro. Ad esempio, si potrebbe creare una stored procedure per generare report personalizzati basati su determinati criteri di ricerca.

3. Database NoSQL: Un database NoSQL (Not Only SQL) è un tipo di database che utilizza modelli di dati diversi rispetto ai tradizionali database relazionali. A differenza dei database relazionali che utilizzano tabelle e schemi fissi, i database NoSQL utilizzano modelli di dati flessibili come documenti, colonne, grafi o chiave-valore. Sono progettati per gestire grandi quantità di dati non strutturati o semistrutturati in modo scalabile e distribuito. Ad esempio, MongoDB è un popolare database NoSQL basato su documenti che memorizza i dati in documenti JSON.

In conclusione, i trigger vengono utilizzati per eseguire azioni specifiche in risposta agli eventi nel database, le stored procedure consentono di eseguire operazioni complesse nel database e i database NoSQL offrono un'alternativa flessibile e scalabile ai tradizionali database relazionali per la gestione di dati non strutturati.